SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Lucija Marković

Zagreb, 2017. godina.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor: Student:

Prof. dr. sc. Mario Essert, dipl. ing. Lucija Marković

Zagreb, 2017. godina.

Izjavljujem da sam ovaj rad izradila samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se svom mentoru prof. dr. sc. Mariu Essertu što mi je omogućio da napišem ovaj rad.

Lucija Marković

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

ZAVRŠNI ZADATAK

Student: **Lucija Marković** Mat. Br.: 0035197362

Naslov rada na hrvatskom jeziku: **Python modul za uporabu Markovljevih lanaca preko Interneta**

Naslov rada na engleskom jeziku: **Python Markov chain module for use over the Internet**

Opis zadatka:

Stohastički proces u kojem uvjetna distribucijska vjerojatnost budućeg stanja procesa, uz zadano sadašnje stanje i sva prošla stanja, ovisi isključivo o sadašnjem stanju, zove se Makovljev proces. Najpoznatiji Markovljevi procesi su Markovljevi lanci, takodjer poznati kao Markovljevi modeli koji predstavljaju diskretne vremenske nizove stanja s prijelaznim vjerojatnostima. Kao poznati primjer Markovljevog procesa ili modela uzima se Brown-ovo gibanje. Različite metode strojnog učenja u modernoj obradbi velikog broja podataka, tzv. 'big data' u različitim područjima medicine i tehnike, prometa, financija i gospodarstva, koriste Markovljeve lance i za njih razvijaju posebne module.

U ovom radu, potrebno je:

1. Upoznati i objasniti matematičku pozadinu Markovljevih procesa
2. Proučiti postojeća programska rješenja Markovljevih lanaca, npr. markovify i PyMarkovChain koja su već načinjena u programskom jeziku Python, ali nemaju mrežne (Internetske) mogućnosti
3. Preinakom postojećih modula, načiniti Python modul i ugraditi ga u web2py mrežni okvir (eng. framework)
4. U mrežnom okviru načiniti odgovarajuće korisničko numeričko i grafičko sučelje
5. Testirati tako dobiveni alat s različitim tehničkim i netehničkim podatcima, preko Interneta

Zadatak zadan:

30. studenog 2016.

Zadatak zadao:

M. Essert

Prof.dr.sc. Mario Essert

Rok predaje rada:
1. rok: 24. veljače 2017.
2. rok (izvanredni): 28. lipnja 2017.
3. rok: 22. rujna 2017.

Predviđeni datumi obrane:
1. rok: 27.2. - 03.03. 2017.
2. rok (izvanredni): 30. 06. 2017.
3. rok: 25.9. - 29. 09. 2017.

v.d. predsjednika Povjerenstva:

Izv. prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	Jedinica	Opis
A		Skup događaja A
B		Skup događaja B
$\mathcal{F}$		Familija podskupova prostora elementarnih događaja
m		Dimenzija matrice
n		Dimenzija matrice, broj ponavljanja
P		Vjerojatnost
$\mathbb{R}$		Skup realnih brojeva
S		Skup stanja Markovljevog lanca
s		Stanje u skupu stanja Markovljevog lanca
T		Skup vremena
t		Svako vrijeme u skupu vremena T
X		Slučajna varijabla
ϖ		Elementarni događaj slučajnog pokusa
Ω		Prostor elementarnih događaja slučajnog pokusa
$\cup$		Unija skupova
$\cap$		Presjek skupova

SAŽETAK

U ovom radu razrađena je primjena Markovljevih lanaca na Internet aplikaciji. Na početku je objašnjena matematička pozadina vjerojatnosti i Markovljevih lanaca. Dan je uvod u Python programerski jezik. Zatim je pojašnjen Markovify, Python paket koji koristi Markovljeve lance za generaciju teksta. Sljedeće je prikazano administrativno sučelje web2py aplikacije. Nakon toga razjašnjeni su detalji novog modula markov.py i prikazana je njegova implementacija na web2py mrežni okvir. Na kraju je prikazano korištenje aplikacije.

Ključne riječi: Markovljev lanc, Python, web2py, modul

SUMMARY

This paper deals with the use of Markov chains on Internet application. The mathematical background of probability and the Markov chains is explained at the beginning. After that, a brief introduction to Python programming language is given. Then, Markovify, a Python package that uses Markov's text generation, is described. Afterwards follows introduction to web2py framework. Next, the details of new module markov.py are clarified and its implementation on the web2py is shown. Lastly, the application's usage is displayed.

Key words: Markov chain, Python, web2py, module

1. UVOD

Ruski matematičar Andrey Andreyevich Markov poznat je po radu na stohastičkim procesima. Primarna tema njegovih istraživanja su Markovljevi lanci i Markovljevi procesi. Temelj Markovljevih lanaca je niz stanja sustava i vjerojatnost prelaska stanja u novo stanje, tj. tranzicija stanja.

Markovljevi lanci imaju široku primjernu. Neka od područja uporabe su: genetsko kodiranje u bioinformatici, prepoznavanje osoba na temelju slike, predviđanje vremenske prognoze, skladanje glazbe, modeliranje različitih procesa u statistici i teoriji redova, itd.

U ovom radu metoda je primijenjena za generiranje teksta. Takvu primjenu možemo pronaći u mobilnim uređajima, kada se prilikom upisivanja riječi predloži sljedeća riječ, ili kod simuliranja ljudskog govora. Korištenjem velikih baza tekstova, generirani tekst trebao bi biti sličan onome koje bi čovjek napisao.

2. MARKOVLJEVI LANCI

2.1. Vjerojatnost

Teorija vjerojatnosti je grana matematike čiji je zadatak formiranje i proučavanje matematičkog modela slučajnog pokusa. Slučajan pokus je takav pokus čiji ishodi nisu jednoznačno određeni uvjetima u kojima je pokus izveden. Elementarni događaj

$$\varpi_1, \varpi_2, \varpi_3, ...\varpi_n \tag{1}$$

je svaki od konačno mnogo ishoda slučajnog pokusa. Neprazan skup

$$\Omega = \{\varpi_1, \varpi_2, \varpi_3, ...\varpi_n\} \tag{2}$$

zove se prostor elementarnih događaja i predstavlja skup svih ishoda slučajnog pokusa.[1] Pretpostavka na prostor elementarnih događanja jest da je neprazan i da sadrži sve što se može realizirati u pokusu.

Vjerojatnost se definira kao mjera skupova (podskupova od Ω). Definira se za dovoljno veliku familiju podskupova od Ω koja čini temelj za definiciju vjerojatnosti. Takva familija naziva se familija događaja. Familija Ƒ podskupova Ω je σ-algebra skupova na Ω ako vjedi:

$$1)\ \Omega \in Ƒ, \varnothing \in Ƒ, \tag{3}$$

$$2)\ A \in Ƒ \rightarrow A^C \in Ƒ, \tag{4}$$

$$3) A, B \in Ƒ \rightarrow A \cup B \in Ƒ . \tag{5}$$

Svojstvo 2) naziva se zatvorenost σ-algebre na komplementiranje (tj. ako sadrži neki skup, sadrži i njegov komplement), a svojstvo 3) zatvorenost σ-algebre na prebrojivu uniju (tj. ako postoji prebrojivo mnogo skupova iz dane σ-algebre, onda će i njihova unija biti element te σ-algebre).[2]

Vjerojatnost je preslikavanje

$$P : F \Rightarrow [0,1] \tag{6}$$

definirano na σ-algebri događaja Ƒ, koje ima svojstva:

1. nenegativnost vjerojatnosti:

$$P(A) \geq 0, A \in Ƒ \tag{7}$$

2. normiranost vjerojatnosti:

$$P(\Omega) = 1 = 1, \tag{8}$$

3. aditivnost vjerojatnosti: ako su A i B disjunktivni događaji, onda je

$$P(A \cup B) = P(A) + P(B). \tag{9}$$

U slučaju da A i B nisu disjunktivni vjerojatnost unije računa se

$$P(A \cup B) = P(A) + P(B) - P(A \cap B), \tag{10}$$

gdje $A \cap B$ označava presjek događaja, tj. događaj koji se ostvaruje ako se ostvare oba događaja A i B.

2.2. Uvjetna vjerojatnost

Vjerojatnost da se dogodio događaj A uz uvjet da je poznato da se dogodio događaj B naziva se uvjetna vjerojatnost događaja A. Traže se elementarni događaji koji čine B i među njima one koji su povoljni za događaj A, tj. traže se elementarni događaji za presjek $A \cap B$ tih događaja.

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}. \tag{11}$$

Događaji A i B su međusobno nezavisni ako ostvarenje događaja B nema utjecaja na vjerojatnost događaja A i vrijedi bilo koja od jednakosti P(A|B)=P(A) ili P(B|A)=P(B). Iz toga slijedi:

$$P(A \mid B) = P(A) = \frac{P(A \cap B)}{P(B)}, \tag{12}$$

$$P(A \cap B) = P(A) \cdot P(B). \tag{13}$$

2.3. Stohastički procesi

Slučajna varijabla je numerički ishod slučajnog eksperimenta. Događajima iz Ω skupine događaja pridružuje se realni broj. Funkcija

$$X : \Omega \Rightarrow \mathbb{R} \tag{14}$$

naziva se slučajna varijabla.

Pojam slučajne varijable neovisan je o vremenu, no poopćuje se tako da uključuje i vremensku komponentu. Promatranje familije slučajnih varijabli koja ovisi o vremenu dovodi do pojma stohastički proces. $T \subset \mathbb{R}$ skup je vremena u kojemu je promatran stohastički proces. Za svako vrijeme $t \in T$ određena je slučajna varijabla koja je označena s X_t. Familija tih slučajnih varijabli definira stohastički proces X:

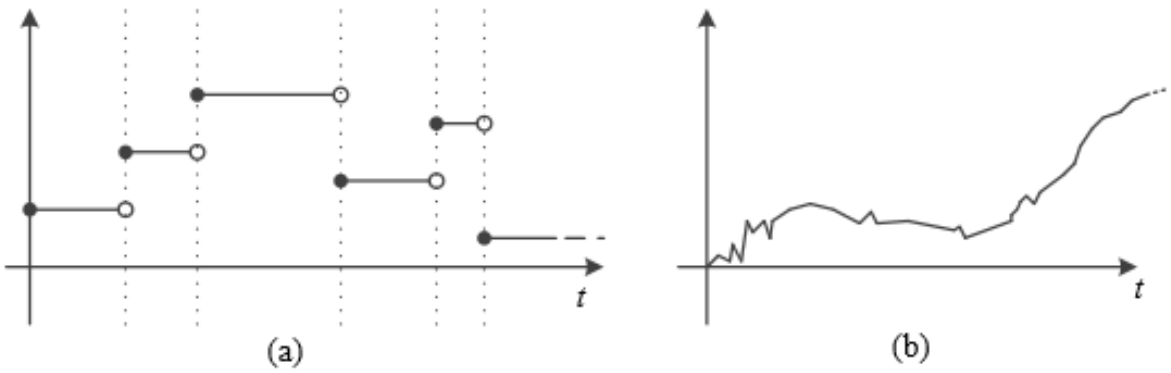
$$X = (X_t, t \in T), \tag{15}$$

gdje je T parametarski skup stohastičkog procesa, a $t \in T$ je parametar.

Stohastički skup može se definirati i kao funkcija dviju varijabli, skupa vremena T i skupa stanja S, skupa unutar kojeg proces poprima vrijednosti

$$X : T \times \Omega \Rightarrow S \,. \tag{16}$$

Stohastičke procese moguće je podijeliti po prirodi skupova T i S. Ukoliko je skup T diskretan, tada je riječ o nizu slučajnih varijabli. Nizove slučajnih varijabli kod kojih je i skup stanja S diskretan proučava teorija Markovljevih lanaca. Kod ostalih stohastičkih procesa vrijeme je kontinuirano. Poissonov proces [Slika 1a] primjer je procesa s kontinuiranim vremenom T i diskretnim skupom stanja S, a Brownovo gibanje [Slika 1b] je proces s kontinuiranim vremenom T i kontinuiranim skupom stanja S.



Slika 1. Stohastički procesi: a) Poissonov proces, b) Brownovo gibanje. [1]

Druga podjela stohastičkih procesa je na stacionarne i nestacionarne. Kod stacionarnih procesa vjerojatnosne osobine invarijantne su u odnosu na pomake vremenskog parametra, dok se one kod nestacionarnih mogu mijenjati.

2.4. Markovljev lanac

Markovljev lanac, nazvan po Andreyu Markovu, predstavlja niz stanja sustava. To je stohastički procesi čije buduće stanje ovisi samo o trenutnom stanju. To svojstvno zove se svojstvo zaboravljivosti. U svakom trenutku sustav može prijeći u novo stanje ili ostati u istom. Promjene stanja nazivaju se tranzicije.

Markovljev lanac je niz diskretnih slučajnih varijabli (X_n, n=0, 1, 2,...). Slučajne varijable uzimaju vrijednosti u konačnom skupo S=$\{s_0, s_1,..., s_n\}$.
Ako za sve izbore stanja $s_0, s_1, s_2, ..., s_n \in S$ vrijedi:

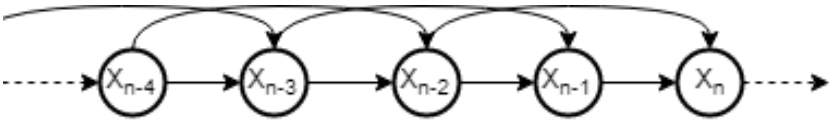
$$P\left(X_n = s_n \mid X_{n-1} = s_{n-1}, ..., X_0 = s_0\right) = P\left(X_n = s_n \mid X_{n-1} = s_{n-1}\right), \tag{17}$$

lanac X_0, X_1,... je Markovljev lanac prvog reda [Slika 2]. Trenutak n predstavlja sadašnjost, a 0,...,n-1 prošlost. Sadašnje stanje s_n ovisi samo o prethodnom s_{n-1}. Sadašnje stanje s_n ne ovisi o vjerojatnostima procesa u ranijim trenutcima, odnosno načinu na koji je proces došao u prethodno stanje.

Slika 2. Markovljev lanac prvog reda

Markovljev lanac drugog reda [Slika 3] ovisi o dvjema prethodnim stanjima s_{n-1}, s_{n-2}. Za njega vrijedi:

$$P\left(X_n = s_n \mid X_{n-1} = s_{n-1},...,X_0 = s_0\right) = P\left(X_n = s_n \mid X_{n-1} = s_{n-1}, X_{n-2} = s_{n-2}\right). \quad \textbf{(18)}$$



Slika 3. Markovljev lanac drugog reda

Prema tome, Markovljev lanac k-tog reda ovisi o k prethodnih stanja s_{n-1}, s_{n-2},..., s_{n-k}.Što je red viši, pamti dalje u prošlost.

Veza između slučajnih varijabli X_n i X_{n-1} zadana je prijelaznim vjerojatnostima. One, za stacionarne Markovljeve lance koji će se ovdje razmatrati, ne ovise o koraku, tj. trenutku. Vjerojatnost prijelaza iz stanja s_i u stanje s_j je

$$p_{n-1} = P\left(X_n = s_j \mid X_{n-1} = s_i\right). \quad \textbf{(19)}$$

Matrica prijelaznih vjerojatnosti označava se s P i u sebi sadrži elemente p_{ij}

$$P = \left(p_{ij}\right) \qquad i, j \in \left\{1,2,3,...,k\right\}. \quad \textbf{(20)}$$

Elementi matrice u nenegativni, a zbroj elemenata u svakom retku jednak je jedan:

$$p_{ij} \geq 0, \qquad \sum_{j=1}^{k} p_{ij} = 1 \qquad \forall i, j \in \left\{1,2,3,...,k\right\}. \quad \textbf{(21)}$$

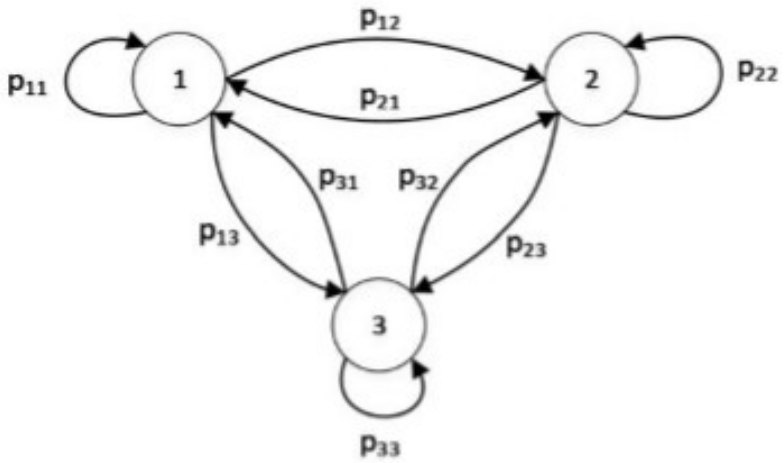
Tako matrica prijelaznih vjerojatnosti Markovljevog lanca s 3 stanja [Slika 4] glasi:

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}, \quad \textbf{(22)}$$

pri čemu:

$$\sum_{j=1}^{3} p_{ij} = 1 \qquad \forall i, j \in \left\{1,2,3\right\}. \quad \textbf{(23)}$$

.



Slika 4. Markovljev lanac s tri stanja [1]

3. PYTHON

Python je programski jezik stvoren kako bi objedinio jednostavnost korištenja, koja postoji kod skriptnih jezika, i visoku funkcionalnost, poput sistemskih jezika. Osmislio ga je 1990. godine Guido van Rossum s ciljem stvaranja koda koji je lako razumljiv radi lako čitljive sintakse. Python je objektno orijentiran jezik. Koristi objekte, odnosno strukture podataka, koji se sastoje od polja podataka i metoda, da bi kreirao program.

U Pythonu je moguć interaktivni i skriptni rad. Skriptni rad sprema se kao datoteka s nastavkom .py. Ta datoteka naziva se modul. Unutar modula moguće je koristite druge module. Poziv modula [Slika 5] vrši se izrazom *import* koji se uobičajeno postavlja na sam početak koda. Python dolazi s bogatom knjižnicom standardnih modula, od kojih su neki ugrađeni u interpreter čime dozvoljavaju korištenje operacija koje nisu u izvornom jeziku.

```
import sys # učitava se modul sistemskih funkcija

from math import sin, cos # učitavaju se samo sin() i cos() funkcije
```

Slika 5. Poziv modula

Paketi izvršavaju sistematizaciju više modula pod zajedničkim imenom. Velika prednost paketa je u razlučivanju sukoba u prostoru imena modula koji se koriste u različitim primjenama. Definiran je imenikom (eng. directory), odnosno mapom, istog imena kao paket. Paket mora sadržavati __init__.py datoteku. Ona ukazuje da imenik sadrži Python paket te dopušta da se paket poziva na isti način kao i modul, izrazom *import*.

Moduli, osim osnovnih izraza i operacija, sadrže funkcije i klase. Funkcija je skupina naredbi koje se izvršavaju po pozivu. Definira se ključnom riječi def i sastoji se od imena funkcije, tj. identifikatora, parametara i naredbi. Naredbom return završava funkcija i dobiva se povratna vrijednost izraza. Funkcija može vratiti i vrijednost *None* ako završava izvršenjem naredbe *return* bez pridruženog izraza ili dolaskom do kraja funkcijskog tijela. Poziv funkcije izvodi se sintaksom *ime_funkcije(argument)*. Pri pozivu funkcije, parametri poziva funkcije povezuju se s funkcijskim argumentima, naredbe unutar funkcije izvršavaju se sa stvarnim vrijednostima argumenata te se nakon izvršenja funkcije vraća određena vrijednosti izraza.

Klasa je Python objekt koji se poziva na isti način kao i funkcija. Atributi klase mogu biti podatčani objekti ili funkcijski objekti. Funkcije unutar klase nazivaju se metode. Metode uvijek imaju obavezan prvi parametar self koji se odnosi na instancu u kojoj se metoda poziva. Instance klase stvaraju se pozivom class objekta. Atributi i metode nastale instance dohvatljivi su korištenjem točka operatora. Unutar klase često je definirana __init__ metoda koja pozivom klase inicijalizira novu instancu. Glavna svrha ove metode je povezivanje atributa novo-nastale instance. Metoda __init__ smije vraćati samo vrijednost *None*, u suprotnom dolazi do *TypeError* iznimke.

3.1. Paket Markovify

Markovify Python paket generator je Markovljevih lanaca, čija je osnovna svrha stvaranje Markovljevih modela iz velike zbirke teksta i generiranje nasumičnih rečenica.
Paket Markovify sastoji se od pet modula:

- __init__.py
- chain.py
- splitters.py
- text. py
- utils.py

3.1.1. __init__.py

Ova datoteka osigurava da Python traži ostale podmodule pri pozivu Markovify paketa. U ovom slučaju __init__.py nije prazna datoteka, već služi kao poziv određenih funkcija iz ostalih podmodula paketa.

```
from .chain import Chain
from .text import Text, NewlineText
from .splitters import split_into_sentences
from .utils import combine
```

Slika 6. __init__.py

3.1.2. chain.py

Datoteka chain.py u sebi sadrži klasu Chain() i funkciju accumulate. Klasa Chain() sadrži funkcije, odnosno metode, koje stvaraju Markovljev lanac za procese koji imaju početak i kraj, u ovom slučaju rečenice. Klasa ima metodu __init__ () koja obavlja inicijalizaciju instanci. Sadrži parametre self, corpus, state_size i model. Corpus je lista lista, gdje je svaka vanjska lista rečenica, a svaka unutarnja lista sadrži riječi rečenica. State-size je broj, integer, koji ukazuje na red modela, tj. koliko riječi ulazi u jedno stanje modela. U generaciji teksta najčešće se koriste modeli drugog ili trećeg reda. Uporabom višeg reda dolazi do citiranja teksta, dok je tekst generiran Markovljevim lancem prvog reda vidljivo besmislen.

```
class Chain(object):
    def __init__(self, corpus, state_size, model=None):
        self.state_size = state_size
        self.model = model or self.build(corpus, self.state_size)
        self.precompute_begin_state()

    def build(self, corpus, state_size):
        if (type(corpus) != list) or (type(corpus[0]) != list):
            raise Exception("`corpus` must be list of lists")
        model = {}
        for run in corpus:
            items = ([ BEGIN ] * state_size) + run + [ END ]
            for i in range(len(run) + 1):
                state = tuple(items[i:i+state_size])
                follow = items[i+state_size]
                if state not in model:
                    model[state] = {}

                if follow not in model[state]:
                    model[state][follow] = 0
```

Slika 7. Klasa Chain()

Build() metoda stvara Markovljev model. Poziv ove metode vraća rječnik rječnika, gdje su sva moguća stanja pohranjena u ključevima vanjskog rječnika. Unutarnji rječnici sadrže sve vjerojatnosti sljedećeg stanja u lancu, zajedno s brojem koliko se puta isto sljedeće stanje pojavilo.

Metoda move() nasumičnim odabirom bira sljedeće stanje s obzirom na trenutno. Unutar metode koristi se modul *bisect* koji održava listu u redoslijedu, bez potrebe da se poziva naredba *sort*.

Gen() metoda generira uzastopna stanja sve dok lanac ne dođe do END stanja. Metoda walk() nadovezuje se na gen() i vraća listu koja sadrži jedno izvšenje Markovljevog modela. Model može biti spremljen kao JSON kako bi se rezultati pohranili u podmemoriju i mogli koristiti kasnije.

```
    def move(self, state):
        if state == tuple([ BEGIN ] * self.state_size):
            choices = self.begin_choices
            cumdist = self.begin_cumdist
        else:
            choices, weights = zip(*self.model[state].items(
            cumdist = list(accumulate(weights))
        r = random.random() * cumdist[-1]
        selection = choices[bisect.bisect(cumdist, r)]
        return selection

    def gen(self, init_state=None):
        state = init_state or (BEGIN,) * self.state_size
        while True:
            next_word = self.move(state)
            if next_word == END: break
            yield next_word
            state = tuple(state[1:]) + (next_word,)

    def walk(self, init_state=None):
        return list(self.gen(init_state))

    def to_json(self):
        return json.dumps(list(self.model.items()))
```

Slika 8. Metode klase Chain()

Funkcija accumulate() u petlji izvršava kumulativno zbrajanje težina, odnosno broja pojavnosti nekog sljedećeg stanja. Koristi se unutar klase Chain prilikom odabira sljedećeg stanja. Ova funkcija koristi modul operator u kojemu su sadržane standardne operacije u obliku funkcija. Accumulate() poziva funkciju operator.add koja je ekvivalent zbrajanju.

```
def accumulate(iterable, func=operator.add):
    it = iter(iterable)
    total = next(it)
    yield total
    for element in it:
        total = func(total, element)
        yield total
```

Slika 9. Funkcija accumulate()

Modul operator, kao i moduli random, bisect i json pozivaju se na samom početku chain modula.

3.1.3. splitters.py

Ovaj modul razdvaja tekst u rečenice uporabom regularnih izraza. Regularni izrazi su niz znakova koji opisuju druge nizove znakova u skladu sa sintaksnim pravilima. Sintaksa se sastoji od literala i metaznakova. Razredi znakova omogućuju raspolaganje s definiranim znakovima unutar skupa. Kako bi izrazi bili čitljiviji, koriste se predefinirani razredi znakova koji su prikazani u sljedećoj tablici:

Tablica 1. Predefinirane klase znakova[8]

Element	Opis	Ekvivalentni razred
.	Odgovara bilo kojem znaku osim newline \n	
\d	Odgovara bilo kojoj decimalnoj znamenki	[0-9]
\D	Odgovara bilo kojem znaku koji nije decimalna znamenka	[^0-9]
\s	Odgovara bilo kojem znaku razmaka	[\t\n\r\f\v]
\S	Odgovara bilo kojem znaku koji nije znak razmaka	[^\t\n\r\f\v]
\w	Odgovara bilo kojem alfanumeričkom znaku	[a-zA-Z0-9]
\W	Odgovara bilo kojem znaku koji nije alfanumerički znak	[^a-zA-Z0-9]

Ponavljanje metaznakova, skupina znakova ili grupa moguće je primijeniti pomoću brojevnih mehanizama, kvantifikatora. Kvantifikatori su prikazani u sljedećoj tablici:

Tablica 2. Kvantifikatori [8]

Simbol	Ime	Kvantifikacija prethodnog znaka
?	upitnik	Uvjetno 0 ili 1 ponavljanje
*	zvjezdica	0, 1 ili više puta
+	plus	Jednom ili više puta
{n,m}	vitičaste zagrade	Između n i m puta
{n}	vitičaste zagrade	Točno n puta
{,n}	vitičaste zagrade	Najviše n puta
{n,}	vitičaste zagrade	Najmanje n puta

Regularni izrazi, kao i svaki modul, pozivaju se naredbom *import*. Najbrže izvođenje regularnih izraza postiže se pretvaranjem u binarni zapis, tj. kompilacijom.

Postoji više metoda pretraživanja. To su match(), search(), findall() i finditer(). Modul splitters.py koristi finditer uzorka [Slika 10] koji traži riječ koja završava s interpunkcijom iza koje može, ali ne mora, slijediti zagrada ili navodnici te razmak.

```
potential_end_pat = re.compile(r"".join([
    r"([\w\.'’&\]\)]+[\.\?!])",
    r"([‘’“”'\"\)\]]*)",
    r"(\s+(?![a-z\-—]))",
    ]), re.U)
```

Slika 10. Uzorak

Finditer() vraća listu sa svim elementima s kojima postoji podudaranje. Svaki element u iteratoru je MatchObject, pa se mogu dobiti grupe koje se podudaraju s uzorkom. Ta opcija upotrebljena je u funkciji split_into_sentences() [Slika 11] . Uzorak se pronalazi u tekstu, te riječ koja završava s interpunkcijom koja pripada grupi 1 prolazi kroz funkciju is_sentence_ender kako bi se dokazalo da nije jedna od iznimaka ili kratica, koje su navedene u modulu.

```
def split_into_sentences(text):
    potential_end_pat = re.compile(r"".join([
        r"([\w\.'’&\]\)]+[\.\?!])", r"([‘’“”'\"\)\]]*)",r"(\s+(?![a-z\-—]))",
        ]), re.U)
    dot_iter = re.finditer(potential_end_pat, text)
    end_indices = [ (x.start() + len(x.group(1)) + len(x.group(2)))
        for x in dot_iter
        if is_sentence_ender(x.group(1)) ]
    spans = zip([None] + end_indices, end_indices + [None])
    sentences = [ text[start:end].strip() for start, end in spans ]
    return sentences
```

Slika 11. Funkcija split_into_sentances()

Funkcija vraća listu rečenica.

3.1.4. text.py

Modul text.py prihvaća unesen tekst, pretvara ga u rečenice, zatim rečenice pretvara u korpus riječi te ga predaje modulu chain koji vraća Markovljev model. Iz tog modela, text.py stvara nove rečenice.

Modul koristi ostale module paketa, pa ih je potrebno pozvati unutar skripte.

Osnovni parametri su input_text, state_size i chain. Chain se poziva unutar __init__ metode, time povezujući parametre text.py i chain.py modula.

```
class Text(object):
    def __init__(self, input_text, state_size=2, chain=None):

        self.input_text = input_text
        self.state_size = state_size
        runs = list(self.generate_corpus(input_text))
        self.rejoined_text = self.sentence_join(map(self.word_join, runs))
        self.chain = chain or Chain(runs, state_size)
```

Slika 12. Metoda __init__() klase Text

Metoda sentance_split() poziva modul splitters i time pretvara tekst u rečenice. Rezultat preuzima metoda generate_corpus(). Ona poziva metodu word_split i time rečenice pretvara u riječi, tj. vrača listu lista riječi.

```
    def sentence_split(self, text):
        return split_into_sentences(text)

    word_split_pattern = re.compile(r"\s+")
    def word_split(self, sentence):
        return re.split(self.word_split_pattern, sentence)

    def generate_corpus(self, text):
        sentences = self.sentence_split(text)
        passing = filter(self.test_sentence_input, sentences)
        runs = map(self.word_split, passing)
        return runs
```

Slika 13. Pretvorba teksta u rečenice i riječi

Metoda make_sentance() [Slika 14] tvori rečenice od riječi dobivenih u Markovljevom modelu. Postoje određeni uvjeti koje stvorena rečenica mora zadovoljiti. Odbacuju se one generirane rečenice koje su identične ili vrlo slične originalnom tekstu, što se određuje maksimalnim udjelom preklapanja, koji je parametar metode test_sentance_output().

```
    def test_sentence_output(self, words, max_overlap_ratio, max_overlap_total):
        overlap_ratio = int(round(max_overlap_ratio * len(words)))
        overlap_max = min(max_overlap_total, overlap_ratio)
        overlap_over = overlap_max + 1
        gram_count = max((len(words) - overlap_max), 1)
        grams = [ words[i:i+overlap_over] for i in range(gram_count) ]
        for g in grams:
            gram_joined = self.word_join(g)
            if gram_joined in self.rejoined_text:
                return False
        return True

    def make_sentence(self, init_state=None, **kwargs):
        tries = kwargs.get('tries', DEFAULT_TRIES)
        mor = kwargs.get('max_overlap_ratio', DEFAULT_MAX_OVERLAP_RATIO)
        mot = kwargs.get('max_overlap_total', DEFAULT_MAX_OVERLAP_TOTAL)

        for _ in range(tries):
            if init_state != None:
                if init_state[0] == BEGIN:
                    prefix = list(init_state[1:])
                else:
                    prefix = list(init_state)
            else:
                prefix = []
            words = prefix + self.chain.walk(init_state)
            if self.test_sentence_output(words, mor, mot):
                return self.word_join(words)
        return None
```

Slika 14. Generiranje rečenica

Još jedna od metoda klase Text jer make_short_sentences() koja nasljeđuje metodu make_sentences(), no ograničena je na određeni broj znakova u rečenici. Sve rečenice iznad ograničenja odbacuje.

3.1.5. utils.py

Modul funkcijom combine() omogućava spajanje više Markovljevih lanaca u jedan. Funkcija traži dva argumenta: modele i težine odnosa modela unutar novog modela.

3.1.6. Primjer korištenja Markovify paketa

U skripti je potrebno izvršiti poziv paketa naredbom *import*. Zatim se otvara tekstualna datoteka pomoću ugrađene funkcije *open(name,[read])*. Dohvaćanjem klase Text() iz modula stvara se objekt koji stvara model. Za načinjenu instancu dohvatljive su metode poput make_sentence() ili make_short_sentence(). Prikazan [Slika 16] je tekst dobiven pozivom metode make_sentance(). Tekst sadrži pet novih rečenica jer se metoda pozvala u petlji.

```
import markovify

with open("/path/to/my/corpus.txt") as f:
    text = f.read()

text_model = markovify.Text(text)

for i in range(5):
    print(text_model.make_sentence())
```

Slika 15. Uporaba paketa Markovify

```
>>>
Within there was a close thing, but I had never so much as their master.
But I see the deadly urgency of this very man McCarthy.
Lord St. Simon is a very coarse one and was left me by the story to which I crouched.
Swiftly I threw myself through, and lay down upon the few acres of ground, and the air of a
 dissolute and wasteful disposition, and the fierce energy of his father, was at least had the
 hardihood to return to England a tomboy, with a massive, strongly marked face and a pair of
 very penetrating dark eyes, and the panelling of the hand that lay upon the sideboard, and
 tearing a piece of white satin shoes and a drunken-looking groom, ill-kempt and side
-whiskered, with an inflamed face and shaking his sides.
As to Mrs. St. Clair through the streets of the Theological College of St. George's, Hanover
 Square, that only half a mile from the shelf one of these last which was passing.
```

Slika 16. Dobiveni rezultat

4. Web2py

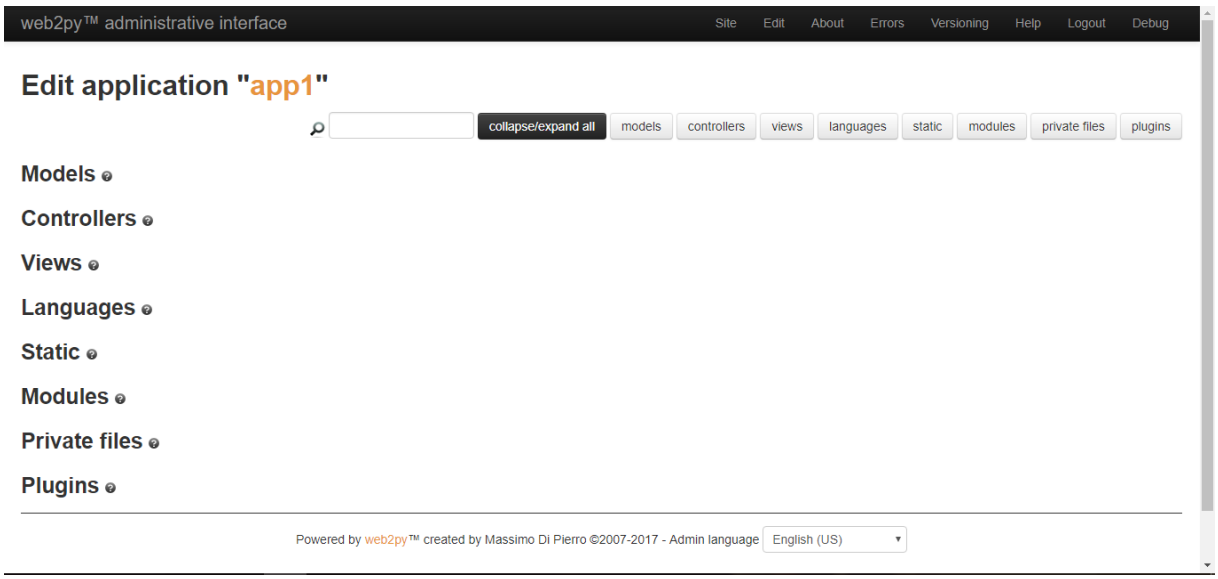
Web2py besplatan je mrežni okvir za razvoj mrežnih aplikacija. Program je pisan i programira se u Python programskom jeziku. Koristi MVC (Model-View-Controller) obrazac softverske arhitekture. Time razdvaja baze podataka, u model-u, od prikaza podataka, unutar view-a, i logike aplikacije i tijeka rada sadržanih u controller-u.
Pokretanjem web2py aplikacije otvara se GUI prozor za odabir servera, koji može biti lokalan ili javan. Korisnik upisuje lozinku koja ga čini administratorom aplikacije i nakon toga se pokreće server.

Unutar administrativnog sučelja nalaze se web aplikacije. Administrator ima mogućnosti stvoriti novu aplikaciju ili unijeti aplikaciju s računala. Aplikacije je moguće urediti, izbrisati, pregledati greške unutar nje, preuzeti, kompilirati i preuzeti kompiliranu, ako je potrebna distribucija aplikacije bez uvida u kod.

Najvažnija radnja je uređivanje. Unutar sučelja za uređivanje nalazi se osnovna struktura [Slika 17] svake web2py aplikacije koju čine:

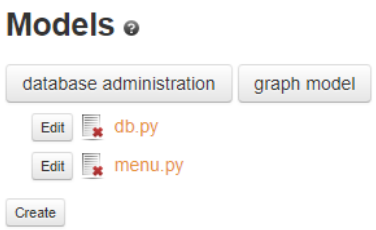
- modeli
- kontroleri
- prikazi
- jezici
- statične datoteke
- moduli
- privatne datoteke
- plugins

Navedeni pojmovi su mape koje sadrže datoteke koje su uredive od strane razvijača aplikacije.



Slika 17. Struktura web2py aplikacije

Modeli su datoteke koje sadrže informacije o konfiguraciji i opise tablica pohranjenih u baze podataka, tj. modeli su opis podataka koji su sadržani u aplikaciji.

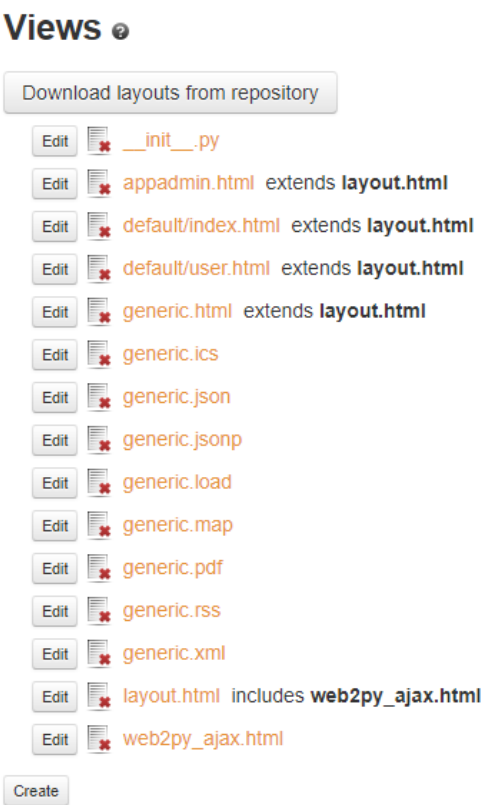


Slika 18. Modeli

Kontroleri sadrže Python module koji definiraju funkcije koje su vezane za web stranice. Svaki URL je mapiran u poziv funkcije unutar kontrolera. Poziv funkcije određuje koje stranice postoje, što se prikazuje na stranicama i kojim podaci stranica može pristupiti, odnosno određuje tijek rada aplikacije.

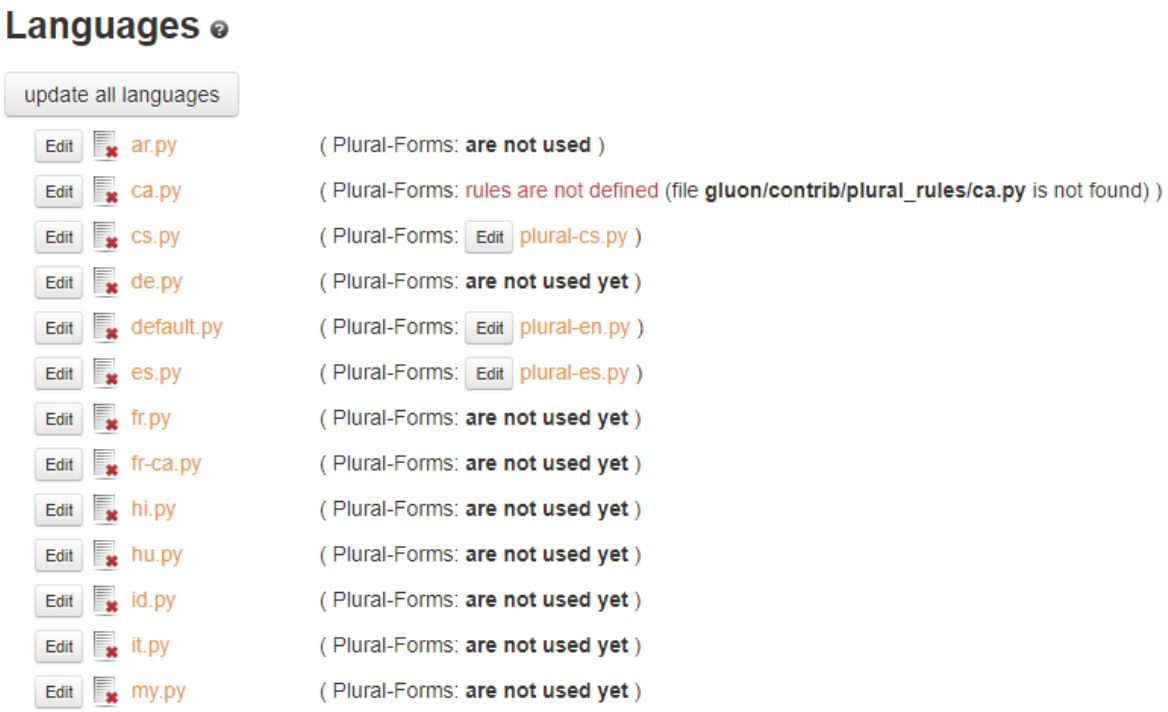
Slika 19. Kontroleri

Prikazi su datoteke koje određuju na koji način su podaci koje prosljeđuje kontroler prikazani. Podaci mogu biti prikazani u HTML-u, XML-u ili u json-u.



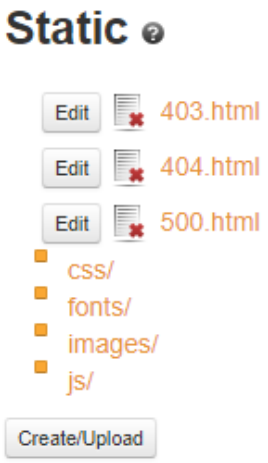
Slika 20. Prikazi

Jezici su datoteke koje sadrže prijevode tekstova unutar aplikacije na različitim jezicima. Ako netko pristupi aplikaciji, čiji je osnovni jezik pretraživača engleski, aplikacija bi trebala biti na engleskom. Ako je osnovni jezik pretraživača njemački, aplikacija bi trebala prevesti stringove na njemački jezik.



Slika 21. Jezici

CSS, slike, medijske datoteke, JavaScript datoteke su statičke datoteke. Te datoteke jednom kada su načinjene, ne trebaju daljnje modificiranje.

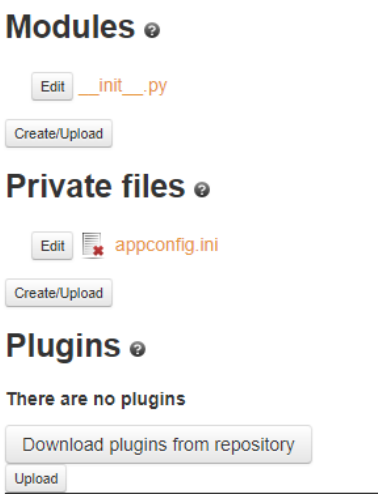


Slika 22. Statične datoteke

Moduli omogućuju korisniku da unese vlastite Python module u aplikaciju.

U privatne datoteke programer unosi datoteke koje su dokučive aplikaciji, ali nisu direktno raspoložive korisnicima web aplikacije.

Plugins su skupine datoteka vezane za aplikaciju koje su neovisne, ne pripadaju ostalim strukturama, ali mogu izgledati kao neka od struktura.



Slika 23. Moduli, privatne datoteke i plugins

4.1. Primjer izrade web2py web aplikacije

U web2py uključen je Database Abstraction Layer (DAL), aplikacijsko programsko sučelje koje mapira Python objekte u objekte baze podataka. DAL dinamički stvara SQL, računalni jezik za izradu, ažuriranje i brisanje podataka iz baza podataka, tako da ga programer ne mora sam pisati.

DAL objekt predstavlja vezu s bazom, a table tablicu baze podataka. DAL stvara instancu tablice. Unutar tablice nalaze se polja koja sadrže argumente baze podataka.

```
db=DAL('sqlite://storage.db')
db.define_table('slika',
                Field('naziv', 'string'),
                Field('datoteka', 'upload'))
```

Slika 24. Model aplikacije- baza podataka

Akcija index() nalazi se unutar kontroler datoteke default.py. Akcija po izvršenju može vraćati string objekt, rječnik, ili sve lokalne varijable.

U kontroleru se nalazi objekt forme. On zna na koji način koristiti vrijednosti predane formi. SQLFORM služi za tvorbu formi iz postojećih baza podataka.

```
def index():
    form=SQLFORM(db.slika).process()
    if form.accepted:
        response.flash= 'Slika unesena!'
    return locals()
```

Slika 25. Akcija default kontrolera

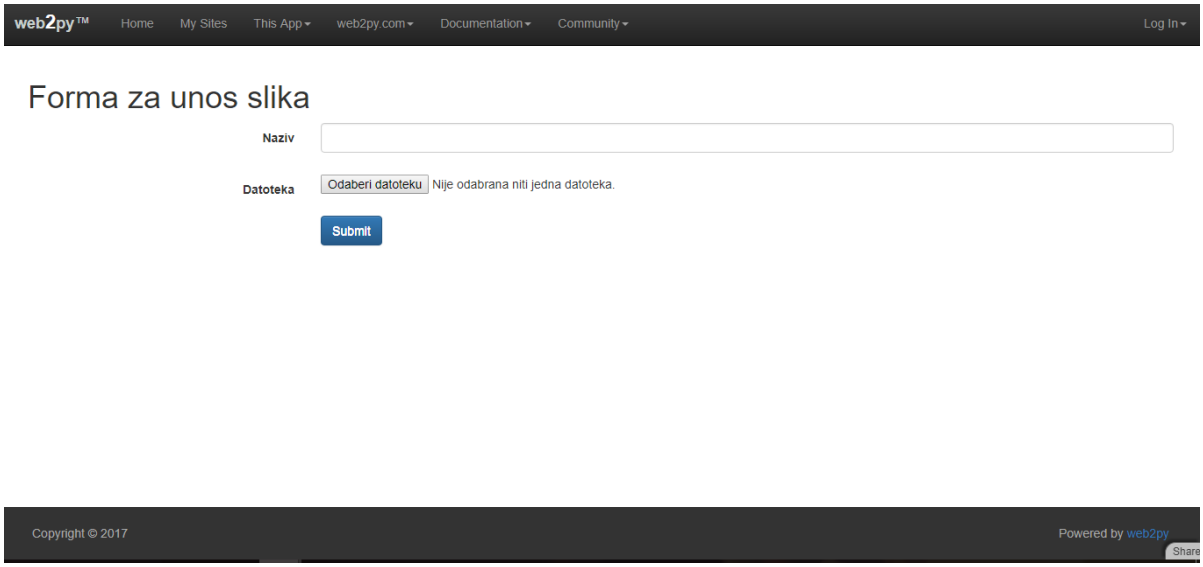
Prikaz koji se veže na akciju mora imati isto ime kao i akcija, u ovom slučaju default\index.html gdje je default ime mape kontrolera, a index akcije. U pregledu se nalazi HTML kod koji definira izgled stranice. Na početku koda poziva se *extend 'layout.html'*, koji sadržiosnovni web2py prikaz stranice. Python funkcije u prikazu upisuju se unutar vitičastih zagrada.

```
{{extend 'layout.html'}}

<h1>
    Forma za unos slika
</h1>
{{=form}}
```

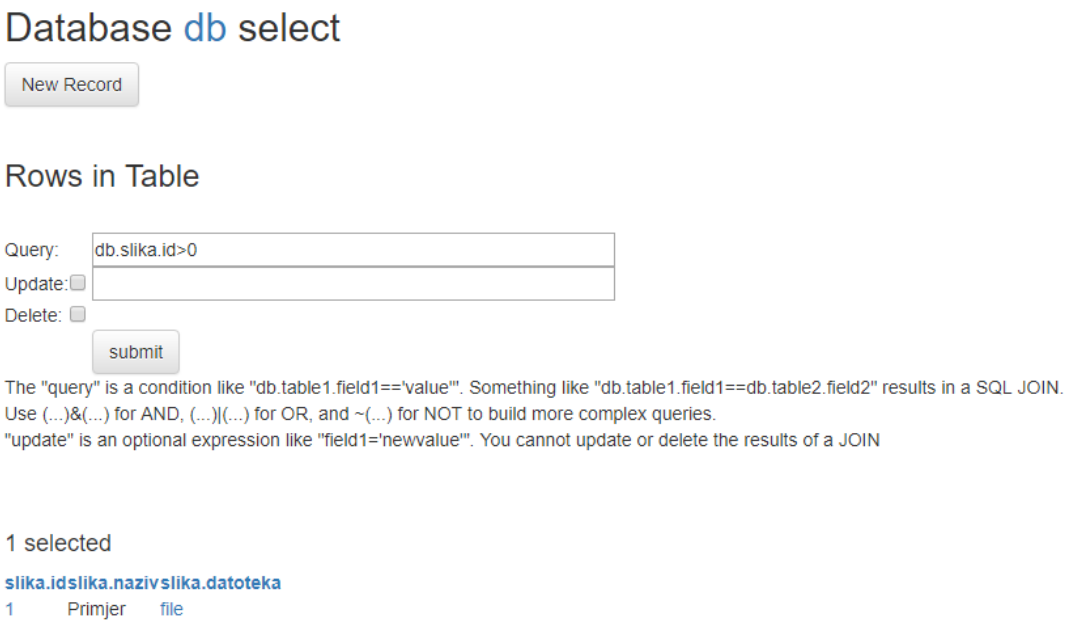
Slika 26. Prikaz aplikacije

Prikaz je realiziran na Slika 27 .

Unesena datoteka nalazi se u bazi podataka i može joj se pristupiti preko administrativnog sučelja za baze [Slika 28].



Slika 27. Sučelje jednostavne aplikacije



Slika 28. Administrativno sučelje baza podataka

5. Modul markov.py

5.1. Python modul

Modul markov.py generator je teksta koji koristi Markovljeve lance. Za razliku od paketa Markovify, gdje su za Markovljev model korištene liste lista, u ovom modulu korišten je matematički zapis rijetkim matricama.

Klasa Markov unutar modula sastoji se od metoda tekst_u_rijeci(), Model(), Recenica(), Sljedece_stanje(), Sljedecih_n_stanja() i Kratki_tekst(). Argument objekta klase kojima se kasnije pozivaju metode je red Markovljevog lanca, kojime korisnik odabire koliko riječi jedno stanje uzima u obzir tvoreći lanac. Tekstualna datoteka koja je odabrana za obradu unosi se kao argument pri pozivu metode tekst_u_rijeci.

5.1.1. Metoda tekst_u_rijeci()

Metoda tekst_u_rijeci() pomoću regularnih izraza razdvaja tekst i stvara listu riječi. Jedna od lista sadrži sve riječi i svako njihovo ponavljanje, odnosno listni je prikaz originalnog teksta, a druga sadrži prvu pojavu svake riječi. Ako je red veći od jedan, elementi tih lista nisu string elementi, već liste s red-brojem elemenata.

```
from scipy.sparse import csr_matrix
from scipy.sparse import spdiags
import numpy as np

class Markov(object):
    def __init__(self,red):
        self.red=red
    def tekst_u_rijeci(self,tekst):
        red=self.red
        clean = re.sub(r'''([\n,',"])''', " ", tekst)
        rijeci = re.split(r"\s", clean.strip())
        rijeci1 = [] # sve pronadene rijeci
        rijeci2 = [] # svaka pronadena rijec/ jedna pojava
        rijeci3 = [] # sve pronadene rijeci u skupovima ovisno o velicini reda
        for r in rijeci:
                rijeci1.append(r)
        for idx in range(0,len(rijeci)):
                    dodaj=rijeci[idx:idx+red]
                    rijeci3.append(dodaj)
        for i in rijeci3:
                if i not in rijeci2:
                    rijeci2.append(i)
        self.l_rijeci =rijeci3#listarijeci viseg reda
        self.l_rijeci_jednom = rijeci2
        self.l_rijeci_sve=rijeci1

        return (self.l_rijeci, self.l_rijeci_jednom,self.l_rijeci_sve)
```

Slika 29. Razlučivanje teksta u riječi modulom markov.py

Liste riječi, pojedinačnih, svih i skupnih, šalju se u metodu Model() koja stvara Markovljev lanac, točnije matricu prijelaznih vjerojatnosti.

5.1.2. Metoda Model()

Model() koristi matrične operacije koje su sadržane u NumPy i SciPy paketima.
NumPy je Python paket koji podržava n-dimenzionalna polja i matrice. Objekt polje (eng. array) fleksibilnija je i numerički efikasnija lista. Za polje vrijede pretpostavke:

- svi elementi moraju biti istog tipa (cijeli, realni ili kompleksni brojevi)
- broj elemenata polja mora biti poznat prilikom njegovog kreiranja

NumPy također sadrži velik broj matematičkih operacija koje se primjenjuju na polja.
Indeksiranje polja, u svrhu dohvaćanja određenog elementa, grupe elemenata ili redaka i stupaca polja, jednako je indeksiranju Python-ovih lista. Prvi element u polju nosi indeks 0, a zadnji M-1, za polje s M brojem elemenata.

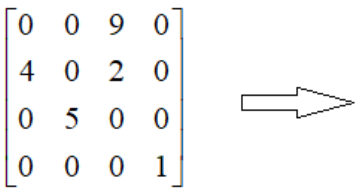
Uz NumPy, primjenjuje se SciPy paket, koji sadrži module za optimizaciju, integraciju, interpolaciju, linearnu algebru, brzu Fourierovu transformaciju, obradu signala i slika, itd. Osnovna struktura podatka koju koristi SciPy je n-dimenzionalno polje iz NumPy-a.

5.1.2.1. Rijetke matrice

Rijetke matrice su one matrice koje imaju vrlo mali broj elemenata različitih od nule. Vrijeme rada s takvim matricama je dugo, u odnosu na mali broj elemenata koji su od interesa. Za pronalazak elemenata matrica dimenzija nxm potrebno je izvršiti nxm broj pretraga. Kako bi se uštedjelo na vremenu obrade podataka i memoriji sustava, rijetke matrice strukturiraju se na drugačiji način. Jedna od struktura rijetkih matrica je oblik s tri stupca:

- redci - indeksi redaka u kojima se nalaze elementi različiti od nule
- stupci - indeksi stupaca u kojima se nalaze elementi različiti od nule
- vrijednosti - iznosi elemenata različitih od nule

Način pretvorbe rijetke matrice u formu tri stupca prikazan je na Slika 30.



Retci	Stupci	Vrijednosti
4	4	21
0	2	9
1	0	4
1	2	2
2	1	5
3	3	1

Slika 30. Rijetka matrica

Sparse (scipy.sparse) jedan je od paketa sadržanih u SciPy paketu. Osnovni zadatak ovog paketa je rad s rijetkim matricama i pripadajućim algoritmima. U Tablica 3 prikazane su klase rijetkim matrica sparse paketa:

Tablica 3. Klase rijetkih matrica

Klasa	Opis
bsr_matrix (arg1, shape, dtype, copy, blocksize])	Block Sparse Row matrica
coo_matrix (arg1, shape, dtype, copy])	Rijetka matrica u COOrdinate formatu
csc_matrix (arg1, shape, dtype, copy])	Compressed Sparse Column matrica
csr_matrix (arg1, shape, dtype, copy])	Compressed Sparse Row matrica
dia_matrix (arg1, shape, dtype, copy])	Rijeka matrica s DIAgonalom memorijom
dok_matrix (arg1, shape, dtype, copy])	Rječnik ključeva baziranih na rijetkoj matrici
lil_matrix (arg1, shape, dtype, copy])	Rijetka matrica u obliku lista vezanih na retke
spmatrix ([maxprint])	Osnovna klasa svih rijetkih matrica

U modulu markov.py korištena je csr_matrix klasa. Instanca te klase može biti načinjena na nekoliko načina:

- csr_matrix(D); gdje je D matrica koja sadrži nula i ne-nula elemente
- csr_matrix(S); gdje je S matrica druga rijetka matrica
- csr_matrix((M,N), [dtype]); za konstruiranje prazne matrice MxN dimenzija
- csr_matrix((data, (row_ind, col_ind)), [shape=(M,N)]); gdje je data lista vrijednosti elemenata koji se nalaze u redcima i stupcima, vezana za liste indeksa redaka i stupaca
- csr_matrix((data, indices, indptr), [shape=(M,N)]); gdje su indeksi stupaca retka i zapisani u indices[indptr[i]]:indptr[i+1] te pripadajuće vrijednosti u data[indptr[i]:indptr[i+1]]

Unutar tijela metode Model() [Slika 31] nalazi se naredba *csr_matrix((data, (idx_r, idx_c)))* koja stvara rijetku matricu. *Idx_r* indeksi su početnih stanja, a *idx_c* indeksi mogućih sljedećih stanja. *Data* je broj pojava sljedećeg stanja nakon početnog stanja. Vrijednosti pojava potrebno je za matricu prijelaznih vjerojatnosti normalizirati, što je učinjeno pomoću NumPy i scipy.sparse funkcija. Metoda traži sve vrijednosti koje imaju zajednički indeks retka, tj. početno stanje im je isto, te zbroj tih vrijednosti dijeli s brojem indeksa stupaca, odnosno brojem mogućih budućih stanja. Poziv metode vraća matricu prijelaznih vjerojatnosti u obliku CSR rijetke matrice.

```
    def Model(self):
        l_rijeci=self.l_rijeci
        l_rijeci_jednom=self.l_rijeci_jednom
        l_rijeci_sve=self.l_rijeci_sve
        red=self.red
        data=[]
        idx_r=[]
        idx_c=[]
        for k in range(0, np.size(l_rijeci_sve)):
                    try:
                        pocetno_stanje = l_rijeci_sve[k:k+red]
                        sljedece_stanje = l_rijeci_sve[k+1:k+red+1]
                        idx_p = l_rijeci_jednom.index(pocetno_stanje)
                        idx_s = l_rijeci_jednom.index(sljedece_stanje)
                        pojava=1.0
                        idx_r.append(idx_p)
                        idx_c.append(idx_s)
                        data.append(pojava)

                    except IndexError:
                        pass
        SM=csr_matrix( (data,(idx_r,idx_c)))
        row_sums = np.array(SM.sum(axis=1))[:,0]
        row_indices, col_indices = SM.nonzero()
        SM.data /= row_sums[row_indices]
        self.NM=SM
        return self.NM
```

Slika 31. Metoda Modelu() za tvorbu normalizirane matrice prijelaznih vjerojatnosti

5.1.3. Metoda Recenica()

Metoda preuzima matricu prijelaznih vjerojatnosti NM te pomoću nje generira rečenicu. Odabire se nasumično početno stanje te se u odnosu na indeks te riječi u listi riječi generira sljedeća riječ. Proces se odvija sve dok sljedeća riječ ne završava s interpunkcijskim znakom, što označava kraj rečenice.

Za odabir sljedećeg stanja [Slika 32] potrebno je iz normalizirane rijetke matrice prijelaznih vjerojatnosti uzeti vrijednosti mogućih budućih stanja u odnosu na indeksirani redak. Te vrijednosti nazvane su težine i pridružene su indeksima budućih stanja. Odabir sljedećeg stanja je nasumičan kako generator ne bi uvijek birao isto stanje. Indeks izabranog stanja sada postaje početni indeks i petlja se ponovno izvršava. Nova riječ nadodaje se listi u kojoj su sadržane do tada izabrane riječi.

```
        pocetak = random.choice(l_rijeci_sve)
        generirana_stanja.append(pocetak)
        if red < 2:
            idx = l_rijeci_jednom.index(pocetak)
        else:
            idx = l_rijeci_sve.index(pocetak)
            generirana_stanja[0]=[pocetak,pocetak]
        while True:
            try:
                uzorak=re.compile("[.?!:]")
                if uzorak.search(pocetak):
                    break
                idx_nr = NM.indices[NM.indptr[idx]:NM.indptr[idx+1]]
                t_redak=NM[idx,:]
                tezine=t_redak.data
                m_lista_tez=zip(idx_nr,tezine)
                rand = random.random()
                suma = 0.0
                for i,t in enumerate(m_lista_tez):
                    suma+=t[1]
                    if rand <= suma:
                        idx = t[0]
                        break
                if red < 2:
                    generirana_stanja.append(l_rijeci_jednom[idx])
                    zadnja_rijec=generirana_stanja[-1]
                else:
                    generirana_stanja.append(l_rijeci[idx])
                    zadnja_rijec=generirana_stanja[-1][0]
                uzorak=re.compile("[.?!:]")
                if uzorak.search(zadnja_rijec[-1]):
                    break
            except IndexError:
                pass
```

Slika 32. Odabir sljedećeg stanja

Poziv metode ispisuje jednu rečenicu sačinjenu od riječi izabranih Markovim lancem pretvorenih u string naredbom ' '.join().

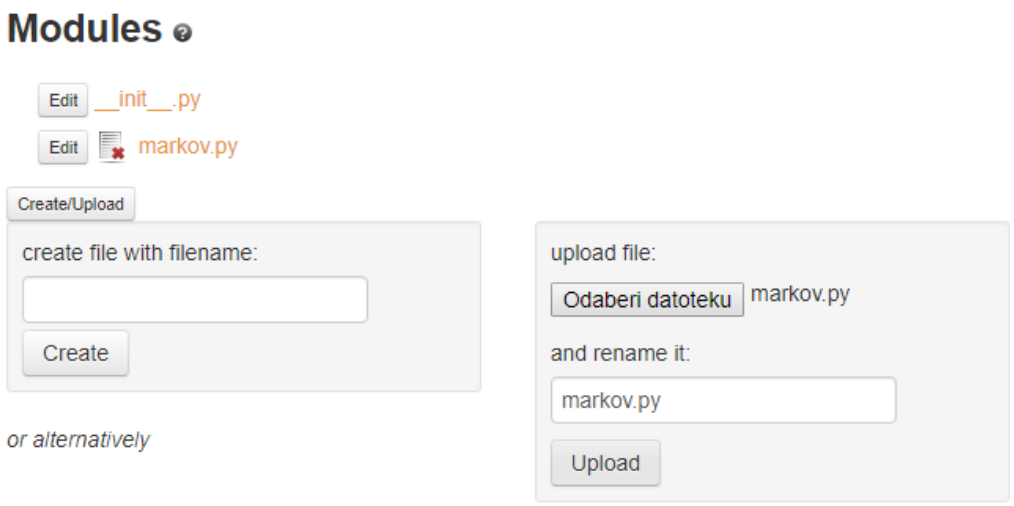
```
        if red < 2:
            recenica =  " ".join(generirana_stanja)
        else:
            generirana_stanja_2=  [item[0] for item in generirana_stanja]
            recenica =" ".join(generirana_stanja_2)
        return recenica[0].upper() + recenica[1:]
```

Slika 33. String generirane rečenice

Metode Sljedece_stanje(), Sljedecih_n_stanja i Kratki_tekst() izvršavaju se na istom principu, ali izlazi su im samo jedno generirano stanje, nekoliko generiranih stanja ili kratki tekst od nekoliko rečenica dobiven korištenjem Recenica() modula u petlji.

5.2. Unos modula u web2py

Modul markov.py unesen je u Model.



Slika 34. Unos markov.py modula

Poziva ga se pomoću naredbe *import markov* unutar default Kontroler datoteke. Za rad modula potrebni su parametri tekst i red Markovljevog lanca. Tekst se čita iz baze podataka tekstova. U Modelu db.py definirana je tablica baze podataka zvana 'datoteka', koja sadrži polja Naslov, Red i tekst.

```
db.define_table('datoteka',
                Field('Naslov','string'),
                Field('Red','integer', default=1),
                Field('tekst','upload'),
                format='%(Naslov)s')
```

Slika 35. Baza 'datoteka'

Baza tekstova puni se na stranici unos_datoteke. Kontrolerska akcija unos_datoteka() sadržava SQLFORM konstruktor koji gradi formu iz tablice baze podataka. U formu korisnik upisuje naslov datoteke, željeni red Markovljevog lanca, kojeg je moguće kasnije promijeniti, i vrši upload datoteke. Predajom forme, baza podataka automatski se obnavlja s unesenim podacima.

```
def unos_datoteke():
    form = SQLFORM(db.datoteka).process(next='unesene_datoteke')
    if form.accepted:
        response.flash='Datoteka unesena. '
    return locals()
```

Slika 36. Kontroler akcija unos_datoteke()

```
<div id="uredi">
    <div id="forma">
        <h1>
            Unos datoteke
        </h1>
        <hr>
        {{=form}}
        <hr>
    </div>
</div>
```

Slika 37. Prikaz datoteka unos_datoteke() akcije

Baza podataka tablično je prikazana na stranici unesene_datoteke() funkcijom *db().select()* čime se biraju svi elementi baze [Slika 38]. Postoji mogućnost promijene reda Markovljevog lanca tipkama + i - . HTML elementi povezani su *onclick* funkcijom koja klikom miša na element pokreće JavaScript skriptu. Klikom funkcija šalje podatke Kontroleru, gdje se u funkciji order_callback() izvršava ažuriranje polja Red baze podataka datoteka. Ako se funkcija izvrši, Prikaz prihvaća nove podatke i prikazuje ih na stranici.

```
def unesene_datoteke():
    rows=db(db.datoteka).select()
    return locals()
```

Slika 38. Kontroler unesene_datoteke()

.

```
<div>
    <h2>Unesene datoteke</h2>
<table class='table' id='uredi'>
    <tr><td>Naziv datoteke</td><td>Red Markovljevog lanca</td></tr>
    {{for row in rows:}}
    <tr  data-id="{{=row.id}}">
        <td ><a href="{{=URL('prikaz',args=row.id)}}">{{=row.Naslov}}</a></td>
        <td >
 <button data-direction="down">-</button><span class='order'>{{=row.Red}}</span>
    <button data-direction="up">+</button></td>
    </tr>
    {{pass}}
</table>
</div>
```

Slika 39. Prikaz datoteka unesene_datoteke() akcije

```
<script>
    function do_ajax_red(t,direction){
        var id = jQuery(t).closest('tr').attr('data-id');
        jQuery.ajax({method : 'get',
                     url : '{{=URL("order_callback")}}',
                     data : {'id':id,'direction':direction},
                     success : function(data){
                         jQuery(t).closest('tr').find('.order').html(data);
                     }});
    }
    jQuery(function(){
           jQuery('[data-direction=up').click(function(){do_ajax_red(this,'up');});
           jQuery('[data-direction=down').click(function(){do_ajax_red(this,'down');})
    });
</script>
```

Slika 40. JavaScript funkcija unutar Prikaz datoteke

```
def order_callback():
    vars = request.get_vars
    if vars:
        id = vars.id
        direction = +1 if vars.direction =="up" else -1
        datoteka = db.datoteka(id)
        if datoteka:
                datoteka.update_record(Red=datoteka.Red+direction)
    return str(datoteka.Red)
```

Slika 41. Pozadinska Kontroler akcija

Izborom datoteke za generaciju teksta, id datoteke dohvaća se iz tablice prikazane u unesene _datoteke() te se kao argument šalje Kontroleru prikaz(). Prikaz() zaprima request.args i odabire zadanu datoteku. U Prikazu se nalaze *button* elementi, tipke, koje nude odabir generiranja rečenice, kratkog teksta ili slijeda stanja. U tim elementima nalaze se reference koje vode korisnika na odabranu funkciju.

```
def prikaz():
    datoteka=db.datoteka(request.args(0))
    return locals()
```

Slika 42. Kontroler akcija prikaz()

```
{{extend 'layout.html'}}
<div>
    <h2>{{=datoteka.Naslov}}</h2>
    </div>
<div class='well' style="width:34%;">
    <h4>{{=A('Generiraj rečenicu',_class="btn btn-
info",_style='width:400px',_href=URL('recenica',args=datoteka.id))}}</h4>
    <h4 >{{=A('Generiraj sljedeće stanje',_class="btn btn-
info",_style='width:400px',_href=URL('sljedece_stanje',args=datoteka.id))}}</h4>
    <h4>{{=A('Generiraj sljedećih nekoliko stanja',_class="btn btn-
info",_style='width:400px',_href=URL('n_stanja',args=datoteka.id))}}</h4>
    <h4> {{=A('Generiraj sažetak',_class="btn btn-
info",_style='width:400px',_href=URL('kratki_tekst',args=datoteka.id))}}</h4>

 </div>
```

Slika 43. Prikaz datoteka prikaz() akcije

Odabirom tipke Generiraj rečenicu, akcija recenica() preuzima id datoteke. Iz baze podataka preuzima tekst i red vezan za datoteku. Poziva se modul markov.py i stvaraju klasni objekti.

```
def recenica():
    datoteka=db.datoteka(request.args(0))
    red=datoteka.Red
    import markov
    import os
    datotekatxt=datoteka.tekst
    tekst= open(os.path.join(request.folder, 'uploads', datotekatxt), 'rb').read()
    text_model = markov.Markov(red)
    lista_rijeci, lista_pojed,lista_rijeci_all=text_model.tekst_u_rijeci(tekst)
    NM=text_model.Model()
    return locals()
```

Slika 44. Kontroler akcija recenica()

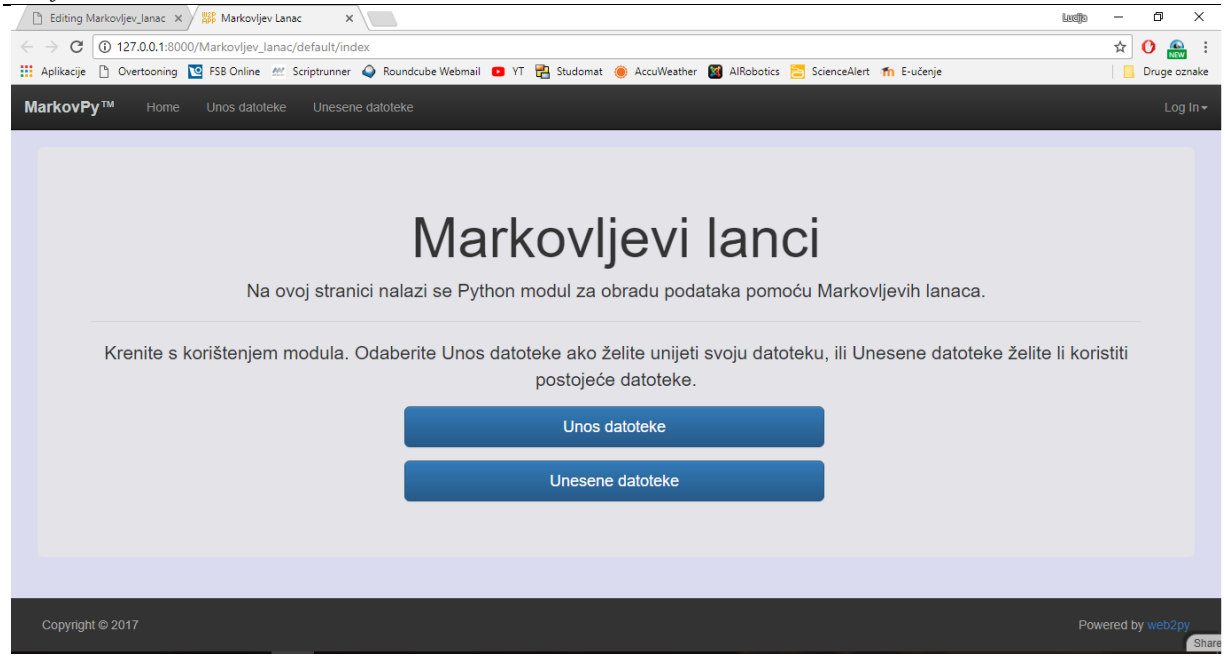
```
{{extend 'layout.html'}}

<div >
    <h2>{{=datoteka.Naslov}}</h2>
    <h3>
        Generirana rečenica: {{=text_model.Recenica()}}
    </h3>
```

Slika 45. Prikaz datoteke recenica() akcije

Izborom kratkog teksta ili slijeda stanja, umjesto {{=text_model.Recenica()}} pozivaju se pripadajuće metode (Kratki_tekst(), Slijed_stanja(), Slijed_n_stanja()).

5.3. Aplikacija iz pogleda korisnika

Početna stranica daje kratko uputstvo o korištenju stranice. Nudi korisniku da unese svoju datoteku ili koristi one koje se već nalaze u bazi podataka. U gornjem dijelu stranice nalazi se izbornička traka koja služi za lakšu navigaciju stranicom.

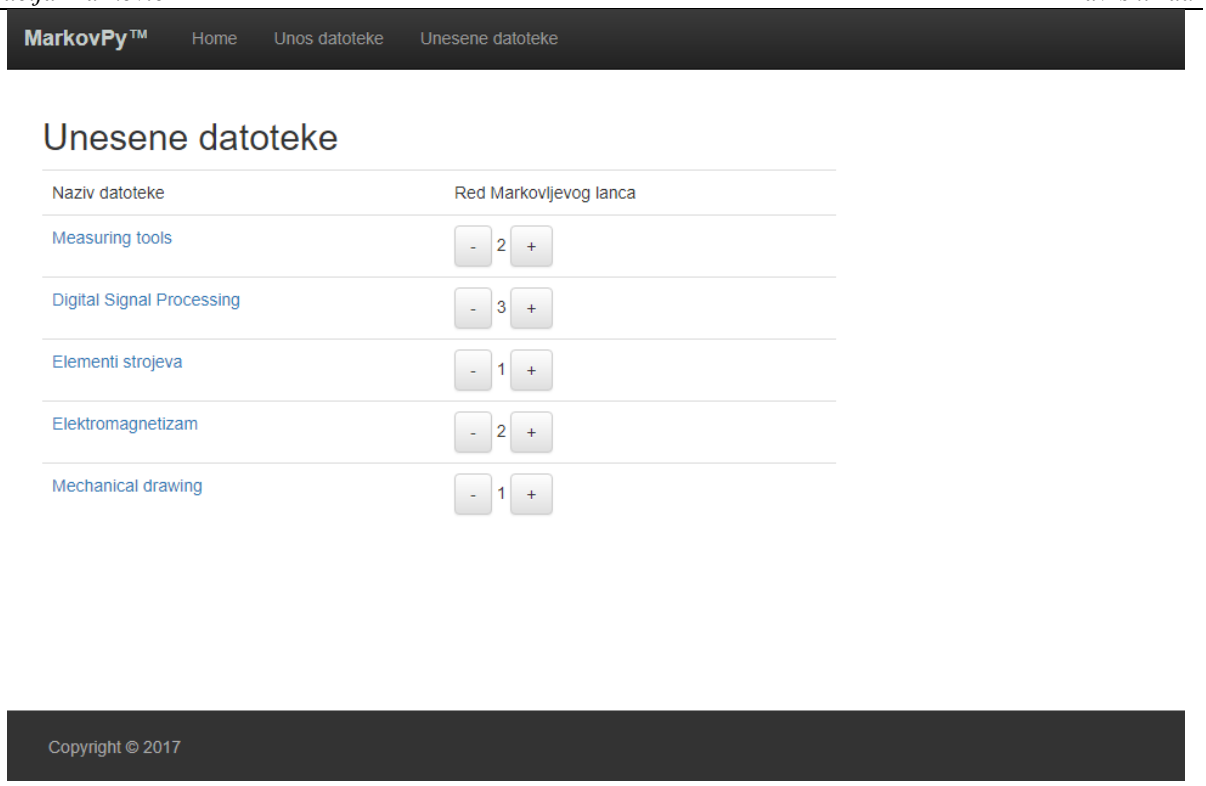


Slika 46. Početna stranica aplikacije

Korisnik upisuje naslov odabrane datoteke u formu. Red je postavljen na 1, no korisnik ga može promijeniti. Predajom forme dolazi do automatskog prijelaza na stranicu s tablicom datoteka.

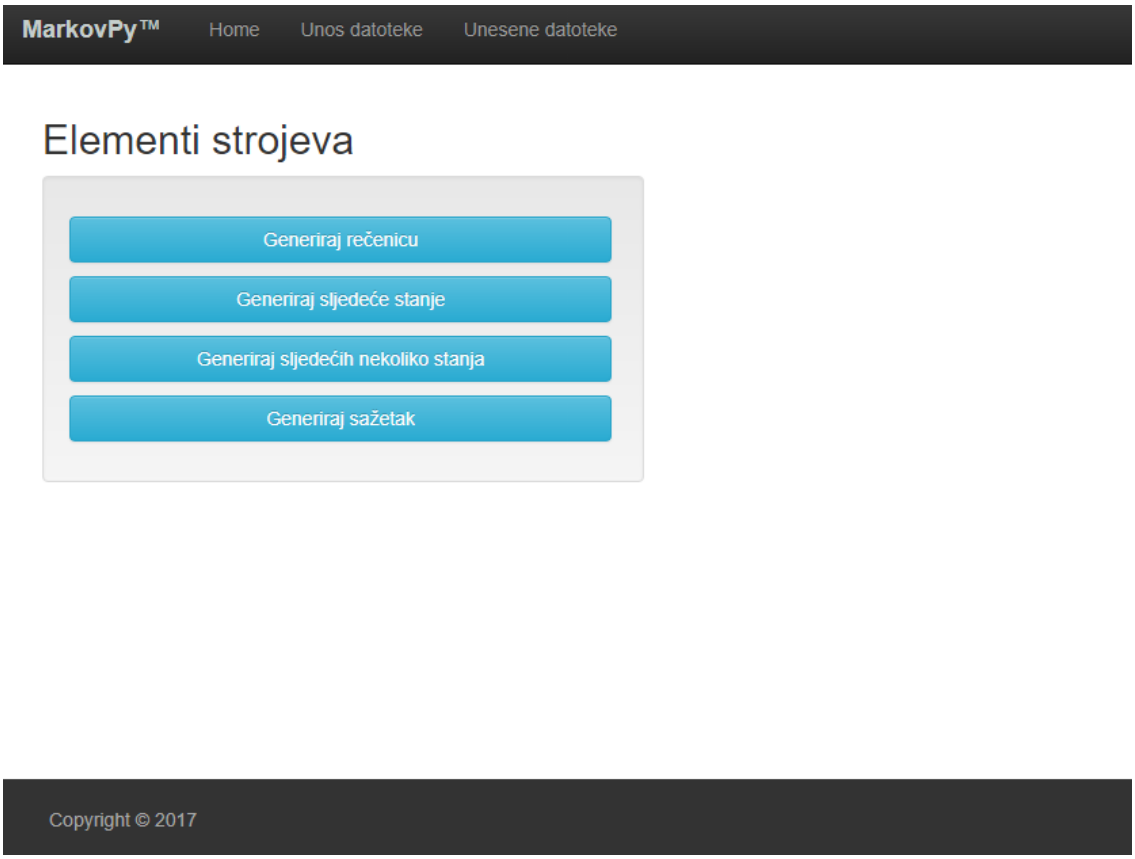
Slika 47. Unos datoteke

U tablici datoteka korisnik može mijenjati red Markovljevih lanaca kako bi uvidio kako promjena reda utječe na rezultate.

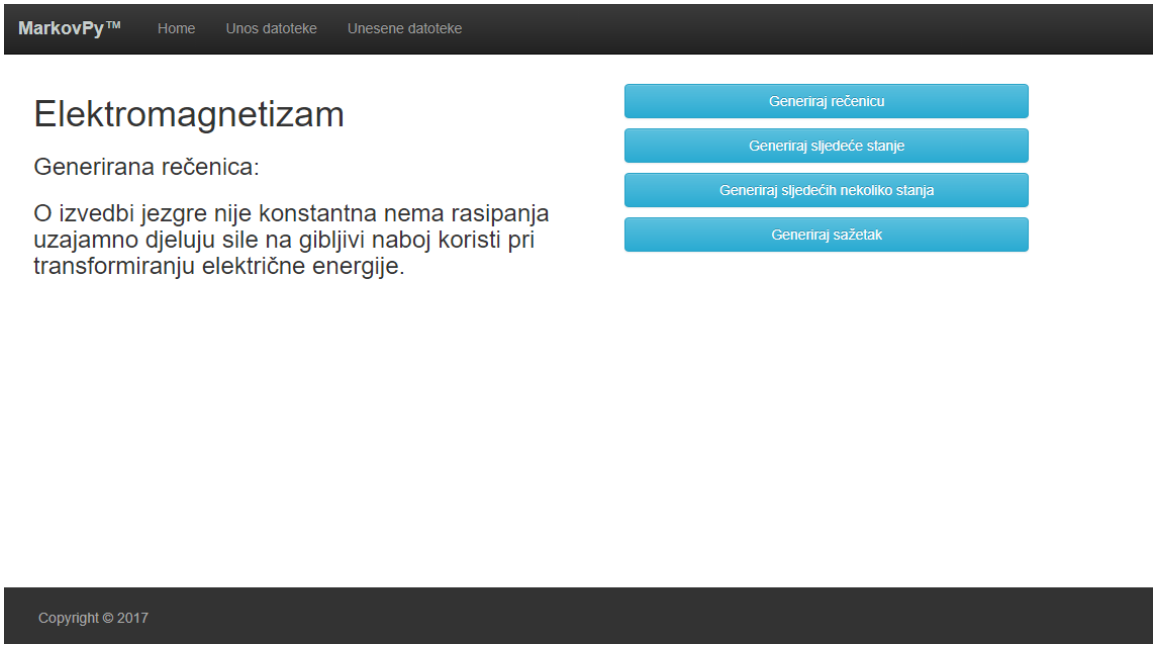


Slika 48. Unesene datoteke

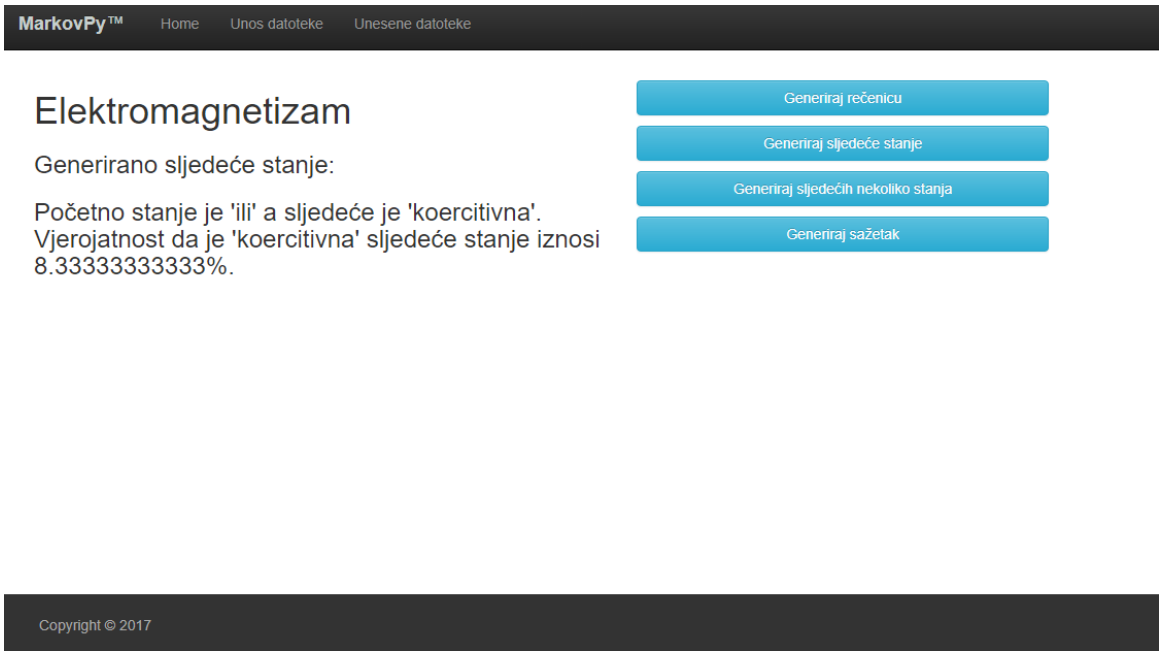
Odabirom željene datoteke, učitava se stranica s prikazom imena datoteke i opcijama za generaciju teksta. Generacija rečenice prikaza je na Slika 50 , a sljedećeg stanja na Slika 51.



Slika 49. Prikaz odabrane datoteke



Slika 50. Generacija rečenice na web aplikaciji



Slika 51. Generacija sljedećeg stanja na web aplikaciji

6. ZAKLJUČAK

Zadatak ovog rada je izrada modula za uporabu Markovljevih lanaca preko interneta. Modul je implementiran u web2py mrežni okvir i vrlo jednostavan za korištenje. U modulu su primijenjene rijetke matrice koje su omogućile rukovanje velikim tekstualnim datotekama bez prekida zbog nedostatka memorije. Bogatijim tekstovima, generirane rečenice postaju sličnije ljuskom izražavanju.

LITERATURA

[1] Šekoranja, B.: Umjetna inteligencija: Markovljevi lanci, Fakultet strojarstva i brodogranje, Zagreb, 2016.

[2] https://www.mathos.unios.hr/uvis/poglavlje1.pdf

[3] http://www.ieee.hr/_download/repository/procesi-05-091117.pdf

[4] https://web.math.pmf.unizg.hr/nastava/opvis/files/opvis_ch3.pdf

[5] https://www.fsb.unizg.hr/usb_frontend/files/1381740398-0-pred1_python.pdf

[6] https://www.fsb.unizg.hr/usb_frontend/files/1478036116-0-regularni_izrazi.pdf

[7] https://www.fsb.unizg.hr/usb_frontend/files/1349774755-0-python_skripta.pdf

[8] http://web2py.com/books/default/chapter/29

PRILOZI

I. CD-R disc